|  |  |
| --- | --- |
| **编号:** | **1-1** |

****

信息科学与工程学院实验报告

《面向对象程序设计》

**Object-Oriented Programming**

|  |  |
| --- | --- |
| 姓名： | 姬彬荃 |
| 学号： | 201911020125 |
| 班级： | 计联培1901 |
| 时间： | 2020年12月2日 |

**《面向对象程序设计》实验报告**

**基本要求：**请围绕实验目的、实验内容、实验过程、实验结果（附图）、实验总结（重点阐述）五个部分进行撰写。若报告中若涉及源代码内容，请在附录部分提供完整源码及GitHub源码托管地址。报告撰写完毕后请提交PDF格式版本到云班课。

1. **实验目的**

* 理解类的三种不同关系（组合，依赖，继承）
* 掌握复合类构造函数、析构函数的定义方法与使用方法
* 熟练掌握类继承的定义方式（单继承，多继承）
* 理解三种不同继承方式间的区别（公有，私有，保护）
* 掌握派生类同名覆盖原理及相应同名冲突解决方法
* 掌握赋值兼容性基本原理（左公 = 右派）
* 熟练掌握复杂类的设计方法（三构一析+普函）

1. **实验内容**

【**任务1**】设计一个基类base，其内含有数据成员（public：int a， protected： int b，private： int c，private：static int count）和函数成员（输出类的数据成员函数print（），统计类对象创建个数的函数 static int statistic（）），然后请采用三种不同的继承方式由base类分别派生出三个子类：derived1，derived2，derived3，请根据上述基类和派生类尝试编程论证下面的三个问题。（可参考课堂演示程序）

* 派生类全盘接受基类的所有本类成员，其中包括基类的普通公有成员，保护成员和私有成员。
* 根据继承类数据成员能否在类内或类外被访问的问题，探索分析三种不同继承方式各自的特点（参考课程ppt）。
* 派生类对象被建立时派生类是如何调用构造函数的，给出构造函数调用的次序，析构函数析构次序，并分析其中规律。

【**任务2**】定义一个二维空间点类Location，采用数据成员x，y表示该类对象在二维坐标系中的坐标位置，类中函数成员函数move（）可以实现移动该类对象的坐标位置，show（）函数可以输出当前类对象的信息。然后，以Location为基类，派生出三维空间坐标点类Point，接着，再利用三维空间点类Point派生出一个三维空间下的球体类Sphere,定义Point点类和球体类Sphere中各自特有的move函数和show()函数。要求设计并实现上述类，并在主函数中定义各个类的对象，通过各自对象调用上述成员函数。

【**任务3**】完成本章所有课后思考与练习题目，给出必要的Visual Studio程序执行结果，解释产生相关问题的原因(写在报告内！)。

1. **实验过程**

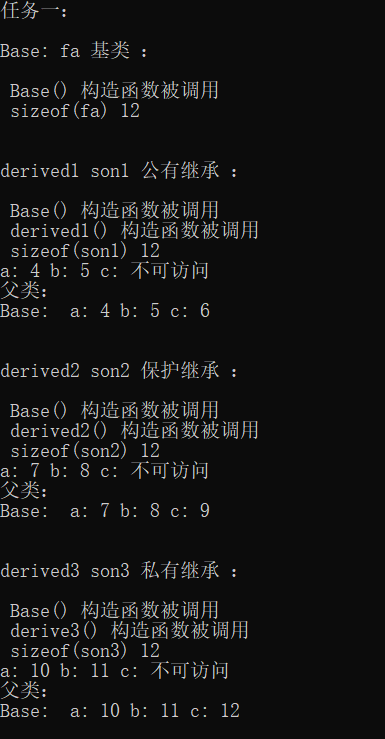
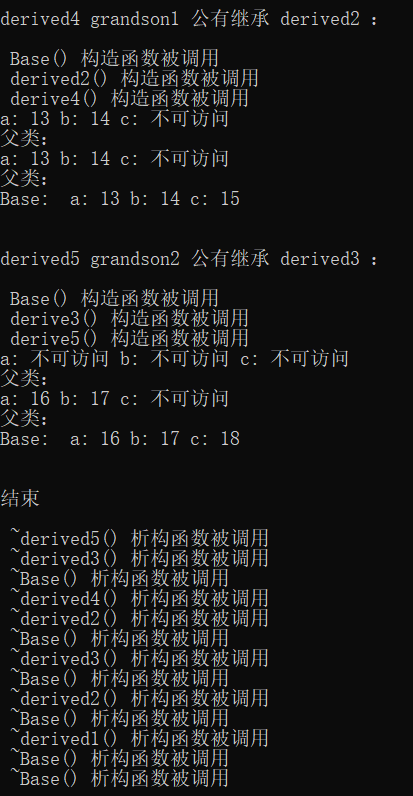
**实验班级**：19计联培 || 19计本 || 19计师本

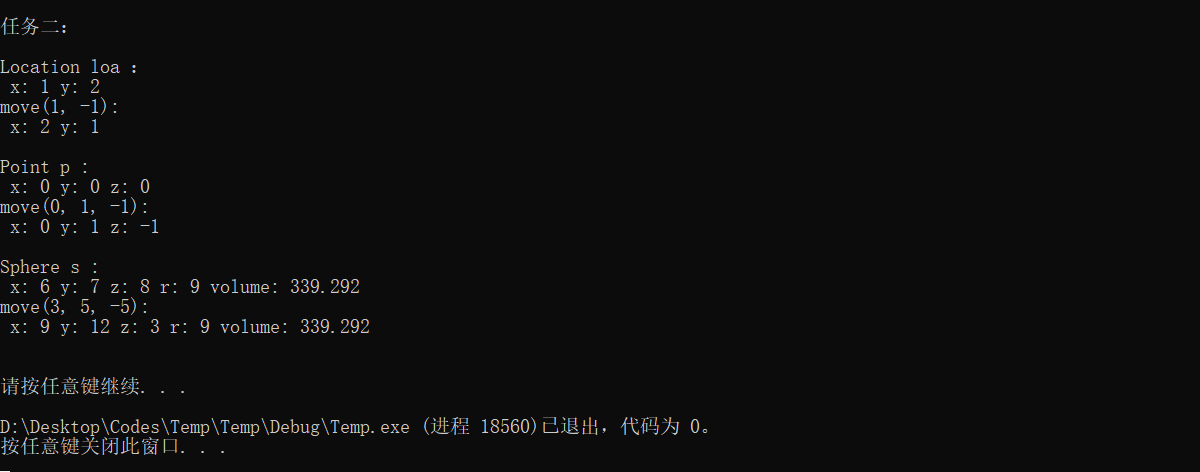
**上机时间**： 2020年11月24日 晚 19:00 -- 20:45

**上机地点**： E315机房 (按组序入座)

**实验环境**： Virtual Studio 2019

1. **实验结果**

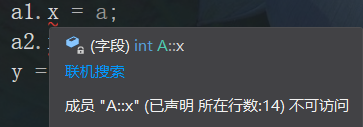
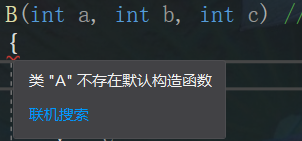


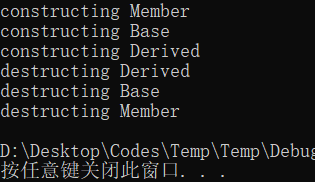


1. **实验总结**

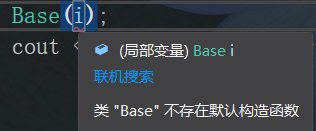
【思考与总结】：

1.（1）不可以，不在初始化列表里调用基类的构造函数系统就会自动调用无参构造函数，而当前A类并没有无参构造函数或含默认参数构造函数。



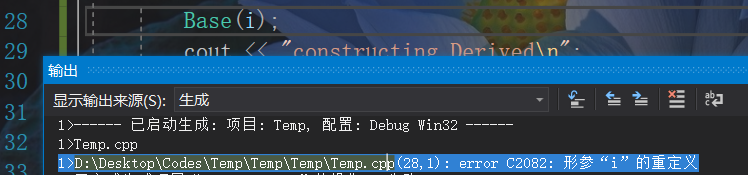
（2）不可以，因为x为A类对象的私有成员，不能直接访问修改。

2.结果如右，因为构造函数的调用顺序是先基类，再成员对象，再自己。

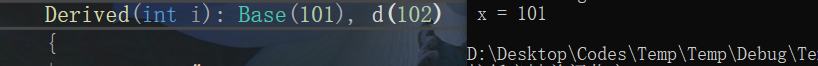
3.(1)不可行，原因同1.(1)。

(2)不可行，VS一样，会认为这里重新新定义了一个形参并将其视作重复定义参数。

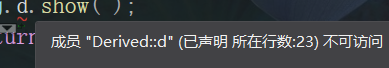
而且这里只是形成了一个临时对象，和本对象的x没有关系



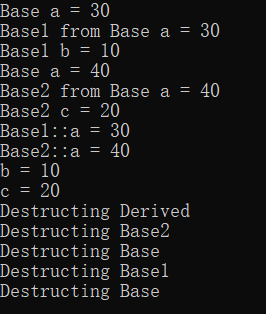
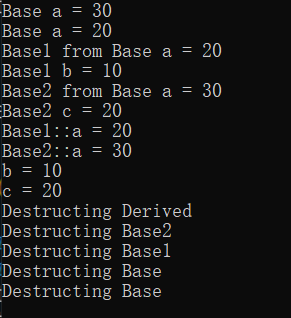
(3)由下图可知，输出的是继承自基类的x。



(4)不可以，d为私有成员对象，不可直接访问。

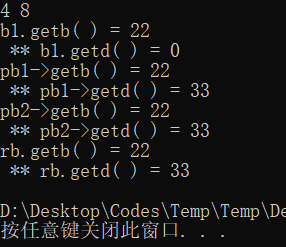


4.运行结果如下：（左为Base2为虚基类，右为Base1为虚基类）



5.变通代码如下：（指针和引用类型可以直接强转，对象由于占内存大小不同，需要放弃多出的数据，通过添加通过基类的构造函数实现强制类型转换）

1. #include <iostream>
2. **using** **namespace** std;
4. **class** Base
5. {
6. **int** b;
7. **public**:
8. Base(**int** x) : b(x)
9. {  }
10. **int** getb()
11. {
12. **return** b;
13. }
14. };
16. **class** Derived : **public** Base
17. {
18. **int** d;
19. **public**:
20. Derived(**int** x, **int** y) : Base(x), d(y)
21. {  }
22. Derived(Base &ba) : Base(ba.getb()){
23. d = 0;
24. }
25. **int** getd()
26. {
27. **return** d;
28. }
29. };
31. **int**  main()
32. {
33. Base b1(11);
34. Derived d1(22, 33);
36. b1 = d1;                                    //第一种赋值兼容
37. cout << **sizeof**(b1) << " " << **sizeof**(d1) << endl;
38. cout << "b1.getb( ) = " << b1.getb() << endl;
39. cout << " \*\* " << "b1.getd( ) = " << Derived(b1).getd( ) << endl;           //不能访问派生类中新增成员
41. Base\* pb1 = &d1;                            //第二种赋值兼容
42. cout << "pb1->getb( ) = " << pb1->getb() << endl;
43. cout << " \*\* " << "pb1->getd( ) = " << ((Derived \* )pb1)->getd( ) << endl;        //不能访问派生类中新增成员
45. Derived\* pd = &d1;
46. Base\* pb2 = pd;                             //第三种赋值兼容
47. cout << "pb2->getb( ) = " << pb2->getb() << endl;
48. cout << " \*\* " << "pb2->getd( ) = " << ((Derived \*)pb2)->getd( ) << endl;     //不能访问派生类中新增成员
50. Base& rb = d1;                              //第四种赋值兼容
51. cout << "rb.getb( ) = " << rb.getb() << endl;
52. cout << " \*\* " << "rb.getd( ) = " << ((Derived &)rb).getd( ) << endl;           //不能访问派生类中新增成员
54. **return** 0;
55. }

结果：

* **附录：程序源码（建议基于Highlight软件导入）**



1. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. \*\* 功能 : 第五章实验（类与类的关系）
3. \*\* 作者 : 2019Jibinquan
4. \*\* 创建 : 2020-12-1 / 19:06
5. \*\* 版权 : 431263064@qq.com
6. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
8. #include <iostream>
9. #include <cstdio>
10. #include <cmath>
11. #include <algorithm>
12. #include <cstring>
13. #include <queue>
14. #include <stack>
15. #include <vector>
16. #include <map>
17. #include <set>
18. #define ios ios::sync\_with\_stdio(false);cin.tie(0);cout.tie(0);
19. #define debug(a) cout << #a << " " << a << endl
20. **using** **namespace** std;
21. **typedef** **long** **long** ll;
22. **const** **double** pi = acos(-1);
23. **const** **double** eps = 1e-8;
24. **const** **int** inf = 0x3f3f3f3f;
25. **const** **int** maxn = 100007;//1e5+7
26. **const** ll mod = 1000000007;//1e9+7
28. /\*----------------------------------\*
29. Class Definition
30. \*-----------------------------------\*/
32. **class** Base {
33. **public**:
34. **int** a;
35. **void** print() {
36. cout << "Base: " << " a: " << a << " b: " << b << " c: " << c << endl;
37. }
38. **static** **int** statistic() {
39. **return** count;
40. }
41. Base(**int** \_a = 0, **int** \_b = 0, **int** \_c = 0) {
42. a = \_a;
43. b = \_b;
44. c = \_c;
45. cout << " Base() 构造函数被调用 " << endl;
46. count++;
47. }
48. ~Base() {
49. cout << " ~Base() 析构函数被调用 " << endl;
50. }
51. **protected**:
52. **int** b;
53. **private**:
54. **int** c;
55. **static** **int** count;
56. };
58. **int** Base::count = 0;
60. **class** derived1 : **public** Base {
61. **public**:
62. **void** show() {
63. cout << "a: " << a << " b: " << b << " c: 不可访问" << endl;
64. cout << "父类：" << endl;
65. Base::print();
66. }
67. derived1(**int** \_a, **int** \_b, **int** \_c) : Base(\_a, \_b, \_c) {
68. cout << " derived1() 构造函数被调用 " << endl;
69. }
70. ~derived1() {
71. cout << " ~derived1() 析构函数被调用 " << endl;
72. }
73. };
75. **class** derived2 : **protected** Base {
76. **public**:
77. **void** show() {
78. cout << "a: " << a << " b: " << b << " c: 不可访问" << endl;
79. cout << "父类：" << endl;
80. Base::print();
81. }
82. derived2(**int** \_a, **int** \_b, **int** \_c) : Base(\_a, \_b, \_c) {
83. cout << " derived2() 构造函数被调用 " << endl;
84. }
85. ~derived2() {
86. cout << " ~derived2() 析构函数被调用 " << endl;
87. }
88. };
90. **class** derived3 : **private** Base {
91. **public**:
92. **void** show() {
93. cout << "a: " << a << " b: " << b << " c: 不可访问" << endl;
94. cout << "父类：" << endl;
95. Base::print();
96. }
97. derived3(**int** \_a, **int** \_b, **int** \_c) : Base(\_a, \_b, \_c) {
98. cout << " derive3() 构造函数被调用 " << endl;
99. }
100. ~derived3() {
101. cout << " ~derived3() 析构函数被调用 " << endl;
102. }
103. };
105. **class** derived4 : **public**  derived2 {
106. **public**:
107. **void** show() {
108. cout << "a: " << a << " b: " << b << " c: 不可访问" << endl;
109. cout << "父类：" << endl;
110. derived2::show();
111. }
112. derived4(**int** \_a, **int** \_b, **int** \_c) : derived2(\_a, \_b, \_c){
113. cout << " derive4() 构造函数被调用 " << endl;
114. }
115. ~derived4() {
116. cout << " ~derived4() 析构函数被调用 " << endl;
117. }
118. };
120. **class** derived5 : **public**  derived3 {
121. **public**:
122. **void** show() {
123. cout << "a: 不可访问" << " b: 不可访问" << " c: 不可访问" << endl;
124. cout << "父类：" << endl;
125. derived3::show();
126. }
127. derived5(**int** \_a, **int** \_b, **int** \_c) : derived3(\_a, \_b, \_c) {
128. cout << " derive5() 构造函数被调用 " << endl;
129. }
130. ~derived5() {
131. cout << " ~derived5() 析构函数被调用 " << endl;
132. }
133. };
135. **class** Location {
136. **protected**:
137. **int** x, y;
138. **public**:
139. **void** trans(**int** \_x, **int** \_y) {
140. x = \_x;
141. y = \_y;
142. }
143. **void** move(**int** dx, **int** dy) {
144. x += dx;
145. y += dy;
146. }
147. **void** show() {
148. cout << " x: " << x << " y: " << y << endl;
149. }
150. Location(**int** \_x = 0, **int** \_y = 0) {
151. x = \_x;
152. y = \_y;
153. }
154. ~Location() {};
155. };
157. **class** Point : **public** Location {
158. **protected**:
159. **int** z;
160. **public**:
161. **void** trans(**int** \_x, **int** \_y, **int** \_z) {
162. x = \_x;
163. y = \_y;
164. z = \_z;
165. }
166. **void** move(**int** dx, **int** dy, **int** dz) {
167. x += dx;
168. y += dy;
169. z += dz;
170. }
171. **void** show() {
172. cout << " x: " << x << " y: " << y << " z: " << z << endl;
173. }
174. Point(**int** \_x = 0, **int** \_y = 0, **int** \_z = 0) : Location(\_x, \_y) {
175. z = \_z;
176. }
177. ~Point() {};
178. };
180. **class** Sphere : **public** Point {
181. **protected**:
182. **double** r;
183. **public**:
184. **void** move(**int** dx, **int** dy, **int** dz) {
185. x += dx;
186. y += dy;
187. z += dz;
188. }
189. **void** show() {
190. cout << " x: " << x << " y: " << y << " z: " << z << " r: " << r << " volume: " << (4.0 / 3 \* pi \* r \* r) << endl;
191. }
192. Sphere(**int** \_x = 0, **int** \_y = 0, **int** \_z = 0, **double** \_r = 0.0) : Point(\_x, \_y, \_z) {
193. x = \_x;
194. y = \_y;
195. z = \_z;
196. r = \_r;
197. }
198. ~Sphere() {};
199. };
201. /\*----------------------------------\*
202. Main Function
203. \*-----------------------------------\*/
205. **void** Task1()
206. {
207. cout << "Base: fa 基类 ：" << endl << endl;
208. Base fa(1, 2, 3);
209. cout << " sizeof(fa) " << **sizeof**(fa) << endl << endl;
210. cout << endl << "derived1 son1 公有继承 ：" << endl << endl;
211. derived1 son1(4, 5, 6);
212. cout << " sizeof(son1) " << **sizeof**(son1) << endl;
213. son1.show();
214. cout << endl << endl << "derived2 son2 保护继承 ：" << endl << endl;
215. derived2 son2(7, 8, 9);
216. cout << " sizeof(son2) " << **sizeof**(son2) << endl;
217. son2.show();
218. cout << endl << endl << "derived3 son3 私有继承 ：" << endl << endl;
219. derived3 son3(10, 11, 12);
220. cout << " sizeof(son3) " << **sizeof**(son3) << endl;
221. son3.show();
222. cout << endl << endl << "derived4 grandson1 公有继承 derived2 ：" << endl << endl;
223. derived4 grandson1(13, 14, 15);
224. grandson1.show();
225. cout << endl << endl << "derived5 grandson2 公有继承 derived3 ：" << endl << endl;
226. derived5 grandson2(16, 17, 18);
227. grandson2.show();
228. cout << endl << endl << "结束" << endl << endl;
229. }
231. **void** Task2()
232. {
233. cout << "Location loa ：" << endl;
234. Location loa(1, 2);
235. loa.show();
236. cout << "move(1, -1): " << endl;
237. loa.move(1, -1);
238. loa.show();
239. cout << endl;
241. Point p;
242. cout << "Point p : " << endl;
243. p.show();
244. cout << "move(0, 1, -1): " << endl;
245. p.move(0, 1, -1);
246. p.show();
247. cout << endl;
248. cout << "Sphere s : " << endl;
249. Sphere s(6, 7, 8, 9.0);
251. s.show();
252. cout << "move(3, 5, -5): " << endl;
253. s.move(3, 5, -5);
254. s.show();
255. }
257. **int** main()
258. {
259. cout << "任务一：" << endl << endl;
260. Task1();
261. cout << endl << endl;
262. cout << "任务二：" << endl << endl;
263. Task2();
264. cout << endl << endl;
265. system("pause");
266. **return** 0;
267. }

Github地址：